What is claimed is:

1. A compiler that translates a source program into a machine language program, including operation definition information in which operation that corresponds to a machine language instruction specific to a target processor is defined, comprising:

a parser step of analyzing the source program;

an intermediate code conversion step of converting the analyzed source program into intermediate codes;

an optimization step of optimizing the converted intermediate codes; and

a code generation step of converting the optimized intermediate codes into machine language instructions,

wherein the intermediate code conversion step includes:

a detection sub-step of detecting whether or not any of the intermediate codes refer to the operation defined in the operation definition information; and

a substitution sub-step of substituting the intermediate code with a corresponding machine language instruction, when the intermediate code is detected, and

in the optimization step, the intermediate codes are optimized, the intermediate codes including the machine language instruction substituted in the substitution sub-step.

2. The compiler according to Claim 1,

wherein the operation definition information is a header file to be included in the source program,

in the header file, the operation is defined by a class made up of data and a method, and

in the intermediate code conversion step, whether or not any of the intermediate codes refer to the operation is detected by detecting whether or not any of the intermediate codes refer to the class defined by the header file.

3. The compiler according to Claim 2,

wherein the class defines a fixed point type, and

in the detection sub-step, intermediate codes that use the fixed point type data are detected.

4. The compiler according to Claim 3,

wherein the method in the class defines operators targeting the fixed point type data,

in the detection sub-step, the detection is executed based on whether or not a set of the operator and the data type targeting an operation agrees with the definition in the method, and

in the substitution step, an intermediate code whose set of the operator and the data type agrees with the definition is substituted with a corresponding machine language instruction.

5. The compiler according to Claim 2,

wherein the class defines a SIMD type, and

in the detection sub-step, the intermediate code using the SIMD type data is detected.

6. The compiler according to Claim 5,

wherein the method in the class defines the operator targeting the SIMD type data,

in the detection sub-step, the detection is executed based on whether or not a set of the operator and the data type targeting an operation agrees with the definition in the method, and

in the substitution step, an intermediate code whose set of the operator and the data type agrees with the definition is substituted with a corresponding machine language instruction.

7. The compiler according to Claim 2,

wherein the class is associated with one machine language instruction that realizes the corresponding processing, and

in the substitution sub-step, the intermediate code is substituted with one machine language instruction associated with the class.

8. The compiler according to Claim 2,

wherein the class is associated with two or more machine language instructions that realize the corresponding processing, and

in the substitution sub-step, the intermediate codes are substituted with two or more machine language instructions associated with the class.

9. The compiler according to Claim 1,

wherein the operation definition information is the header file included in the source program,

in the header file, the operation is defined by the function, and

in the intermediate code conversion step, whether or not any of the intermediate codes refer to the operation is detected by detecting whether or not any of the intermediate codes refer to the function defined by the header file.

10. The compiler according to Claim 9,

wherein the function describes one machine language instruction that realizes the corresponding processing, and

in the substitution sub-step, the intermediate code is substituted with one machine language instruction described in the function.

11. The compiler according to Claim 10,

wherein the function includes a function that returns the number of bits represented as a series of 0s from the most significant bit of input data, and

a machine language instruction described in the function counts the number of bits represented as a series of 0s from the most significant bit of a value stored in the first register and stores the result in the second register.

12. The compiler according to Claim 10,

wherein the function includes a function that returns the number of bits represented as a series of 1s from the most significant bit, and

a machine language instruction described in the function counts the number of bits represented as a series of 1s from the most significant bit concerning the value stored in the first register and stores the result in the second register.

13. The compiler according to Claim 10,

wherein the function includes a function that returns the number of bits that the same value as the most significant value of input data succeeds, and

a machine language instruction described in the function counts the number of bits represented by a series of the same value as the most significant bit of the value stored in the first register and stores the result in the second register.

14. The compiler according to Claim 13,

wherein the function returns the number of bits represented as a series of the same value as the most significant value of input data from the next bit to the most significant bit, and

a machine language instruction described in the function is counts the number of bits represented as a series of the same value

as the most significant bit from the next bit to the most significant bit of the value stored in the first register and stores the result in the second register.

15. The compiler according to Claim 10,

wherein the function includes a function that returns the number of bits of 1 included in input data, and

a machine language instruction described in the function counts the number of bits of 1 of the value stored in the first register and stores the result in the second register.

16. The compiler according to Claim 10,

wherein the function includes a function that returns an sign-expanded value based on bits extracted at designated bit positions from input data, and

a machine language instruction described in the function takes out bits at the bit positions designated by the second register from the value stored in the first register, sign-expands said bits and stores the sign-expanded bits in the third register.

17. The compiler according to Claim 10,

wherein the function includes a function that returns an zero-expanded value based on bits extracted at designated bit positions from input data, and

a machine language instruction described in the function takes out bits at the bit positions designated by the second register from the value stored in the first register, zero-expands said bits and stores the zero-expanded bits in the third register.

18. The compiler according to Claim 9,

wherein the function describes a machine language instruction sequence including two or more machine language

instructions that realize corresponding processing, and

in the substitution sub-step, the intermediate codes are substituted with the machine language instruction sequence.

19. The compiler according to Claim 18,

wherein the function includes a function that updates an address of modulo addressing.

20. The compiler according to Claim 19,

wherein the machine language instruction sequence described in the function includes a machine language instruction that stores in the third register a value acquired by substituting a predetermined bit field of the value stored in the first register with a value stored in the second register.

21. The compiler according to Claim 18,

wherein the function includes a function that updates an address of bit reverse addressing.

22. The compiler according to Claim 21,

wherein the machine language instruction sequence described in the function includes a machine language instruction that stores in the third register a value acquired by inverting bit by bit a position of a predetermined bit field of the value stored in the first register.

23. The compiler according to Claim 9,

wherein the function includes a function that can designate a temporary variable with the accumulator as a reference type, the function being an operation of updating both an accumulator that does not target allocation in optimization and a general purpose register that targets allocation in optimization.

24. The compiler according to Claim 23,

wherein the function performs a multiplication and can designate a temporary variable with an accumulator as a reference type, the accumulator storing the result of the multiplication.

25. The compiler according to Claim 23,

wherein the function performs a sum of products and can designate a temporary variable with an accumulator as a reference type, the accumulator storing the result of the sum of products.

26. The compiler according to Claim 9,

wherein in the substitution sub-step, the intermediate code referring to the function is substituted with a machine language instruction having a variety of operands corresponding to a variety of arguments of said function.

27. The compiler according to Claim 26,

wherein in the substitution sub-step, an intermediate code referring to the function is substituted with (i) a machine language instruction whose operand is a constant value acquired by holding in constants when all arguments are constants; (ii) a machine language instruction that has an immediate value operand when a part of arguments are constants; and (iii) a machine language instruction that has a register operand when all arguments are variable.

28. The compiler according to Claim 1,

wherein the optimization step includes a type conversion sub-step of substituting a plurality of intermediate codes or machine language instructions that perform an operation between different types with one machine language instruction that performs said

operation.

29. The compiler according to Claim 28,

wherein in the type conversion sub-step, a plurality of intermediate codes or machine language instructions that perform an operation that multiplies two n-bit variants and stores the result in a 2n-bit variant are substituted with one machine language instruction that performs said operation.

30. The compiler according to Claim 29,

wherein in the type conversion sub-step, the operation is substituted with the machine language instruction when an explicit declaration that a type conversion from n bits to 2n bits is carried out is made toward the two variants.

31. The compiler according to Claim 1,

wherein the compiler targets a processor that has two or more fixed point modes of performing an operation targeting two or more fixed point types.

in the parser step, a description that the fixed point mode is switched is detected in the source program, and

the compiler further includes a fixed point mode switch step of inserting a machine language instruction to switch fixed point modes following the description that the fixed point mode is switched, when the description is detected in the parser step.

32. The compiler according to Claim 31,

wherein the description that the fixed point mode is switched is associated with a target function, and

in the fixed point mode switch step, machine language instructions for saving and returning of the fixed point mode are inserted respectively into the head and the tail of a corresponding

function.

33. The compiler according to Claim 1,

wherein the optimization step includes a latency optimization sub-step of detecting a description in the source program, the description designating latency in which execution time at the specific position is secured only for a predetermined number of cycles, and scheduling a machine language instruction so that the latency is secured according to the detected designation.

34. The compiler according to Claim 33,

wherein in the latency optimization sub-step, when a description that latency of a predetermined cycles is designated targeting an interval between a first machine language instruction to which a first label is attached and a second machine language instruction to which a second label is attached is detected, the scheduling is executed in order that it takes execution time of only the number of cycles since the first machine language instruction is executed until the second machine language instruction is executed.

35. The compiler according to Claim 33,

wherein in the latency optimization sub-step, when a description that latency of a predetermined cycles is designated targeting access to a specified register is detected, the scheduling is executed in order that it takes execution time of only the number of cycles since a machine language instruction to access the register is executed until a machine language instruction to access said register is executed next time.

36. The compiler according to Claim 1,

wherein the compiler further comprises a class library to substitute a machine language instruction used in the operation

definition information with a machine language instruction of a second processor that is different from a first processor that said compiler targets.

37. A computer-readable recoding medium on which a header file included in a source program to be compiled is recorded,

wherein operation definition information in which operation that corresponds to a machine language instruction specific to a target processor is defined is a header file included in the source program,

in the header file, the operation is defined by a class made up of data and a method.

38. A computer-readable recoding medium on which a class library included in a source program to be compiled is recorded,

wherein the compiler further comprises a class library to substitute a machine language instruction used in the operation definition information with a machine language instruction of a second processor that is different from a first processor that said compiler targets.

39. A computer-readable recording medium on which a source program to be compiled including at least one of a header file or a class library is recorded,

wherein operation definition information in which operation that corresponds to a machine language instruction specific to a target processor is defined is a header file included in the source program,

in the header file, the operation is defined by a class made up of data and a method, and

the compiler further comprises a class library to substitute a machine language instruction used in the operation definition

information with a machine language instruction of a second processor that is different from a first processor that said compiler targets.

40. A compiler apparatus that translates a source program into a machine language program, the compiler apparatus comprising:

a unit operable to hold operation definition information in which operation that corresponds to a machine language instruction specific to a target processor is defined in advance;

a parser unit operable to analyze the source program;

an intermediate code conversion unit operable to convert the analyzed source program into intermediate codes;

an optimization unit operable to optimize the converted intermediate codes;

a code generation unit operable to convert the optimized intermediate codes into machine language instructions,

wherein the intermediate code conversion unit includes:

a detection unit operable to detect whether or not any one of the intermediate codes that refer to the operation defined in the operation definition information;

a substitution unit operable to substitute an intermediate code with a corresponding machine language instruction, when the intermediate code is detected, and

the optimization unit performs optimization with the intermediate codes including the machine language instruction substituted in the substitution unit.

41. A compilation method for translating a source program into a machine language program comprising,

a parser step of analyzing the source program;

an intermediate code conversion step of converting the analyzed source program into intermediate codes;

an optimization step of optimizing the converted intermediate codes; and

a code generation step of converting the optimized intermediate codes into machine language instructions, and

wherein the intermediate code conversion step includes:

a detection sub-step of detecting whether or not any one of the intermediate code refer to the operation defined in operation definition information in which operation that corresponds to a machine language instruction specific to a target processor is defined in advance;

a substitution sub-step of substituting an intermediate code with a corresponding machine language instruction when the intermediate code is detected, and

in the optimization step, the intermediate codes are optimized, the intermediate codes including the machine language instruction substituted for the intermediate code in the substitution sub-step.